School of Computer Science & Engineering

Trustworthy Systems Group

A Rely-Guarantee + Separation Logic for Verifying Microkit-Based Systems

Local RG over the seL4 trace monad with precise shared-region invariants

Junming Zhao

junming.zhao@unsw.edu.au

PhD Student

Trustworthy Systems, UNSW

Motivation: Verifying LionsOS

Overview:

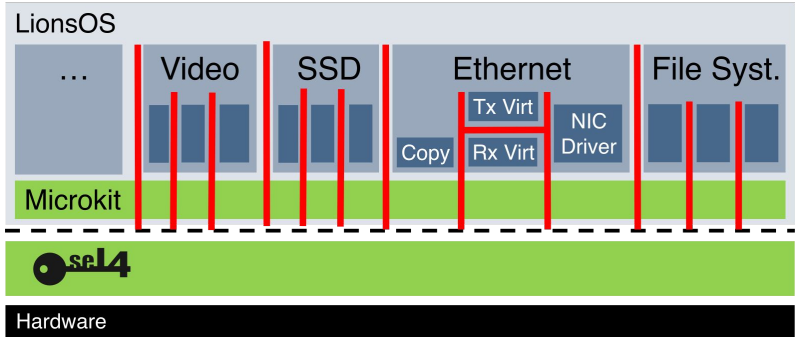
- Verified seL4 + Microkit
- User-level PDs: concurrent
- Need a composition rule for PDs interference

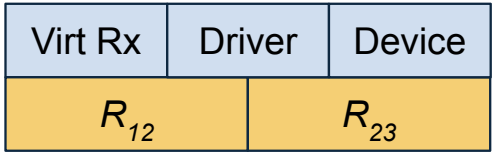
Microkit static structure:

- Fixed PDs
- Fixed memory regions and channels

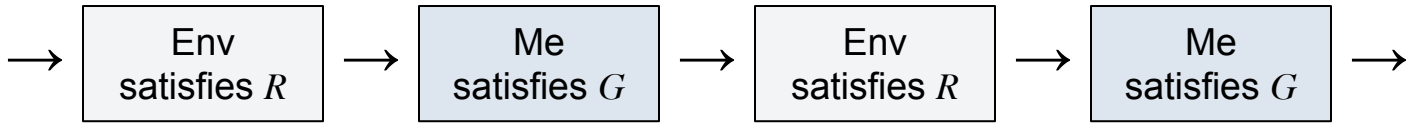
Proof goals:

- Prove each component locally
- Frame it into the system state, then finally
- Compose one global property.





Rely-Guarantee



Rely-Guarantee — $\{P\}\ \{R\}\ f\ \{G\}\ \{Q\}$

- **Pre-condition** (P): Initial state
- **Post-condition** (Q): Final state
- **Rely** (R): Environmental steps
- **Guarantee** (G): Component's own steps

$$\{P_1\}\ \{R_1\}\ f_1\ \{G_1\}\ \{Q_1\}$$

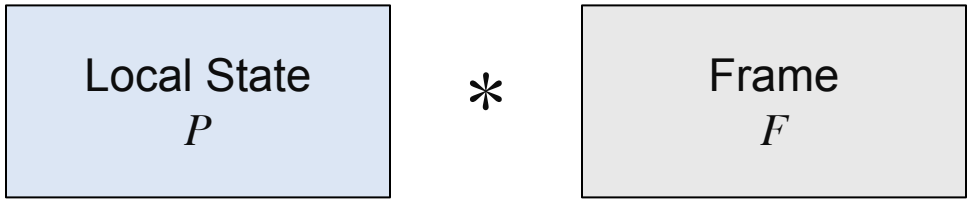
$$\{P_2\}\ \{R_2\}\ f_2\ \{G_2\}\ \{Q_2\}$$

$$\{P_3\}\ \{R_3\}\ f_3\ \{G_3\}\ \{Q_3\}$$

Parallel composition checks compatibility:

$$G_i \leq R_j \quad (i \neq j)$$

Separation Logic



$(P * F)(s)$ iff s splits into disjoint sP and sF,
with $P(sP)$ and $F(sF)$

$$\frac{\{P\}\ f\ \{Q\} \quad f \text{ does not touch } F}{\{P * F\}\ f\ \{Q * F\}}$$

Separation Logic

$$\frac{\{P\}\ \{R\}\ f\ \{G\}\ \{Q\} \quad f \text{ does not “touch” } F}{\{P*F\}\ \{R*F\}\ f\ \{G*F\}\ \{Q*F\}}$$

Trace-Monad Semantics

$$f : \Sigma \to \mathscr{P}(\text{Trace} \times \text{Out}(\alpha))$$

Given $\sigma \in \Sigma$, $f\,\sigma \subseteq \text{Trace} \times \text{Out}(\alpha)$

$$\text{Trace} = (\text{Id} \times \Sigma)^*$$

$$\text{Id} = \{\texttt{Me}, \texttt{Env}\}$$

$$\text{Out}(\alpha) = \{\ \texttt{Failed}, \texttt{Incomplete}, \text{Result}\ (\alpha \times \Sigma)\}$$

R filters Env-labelled steps; G filters Me-labelled steps.

Trace-Monad Semantics

Example Primitives

```
return v = λs.{([], Result(v,s))}
  -- returns v, emits no trace step

modify f = λs.{([], Result((),f s))}
  -- applies modification f to current
state, emits no trace step

commit_step = λs.{([], Incomplete),
([(Me, s)], Result((),s))}
  -- emits one Me step

modify_commit f = modify f; commit_step
  -- one committed Me-labelled update
```

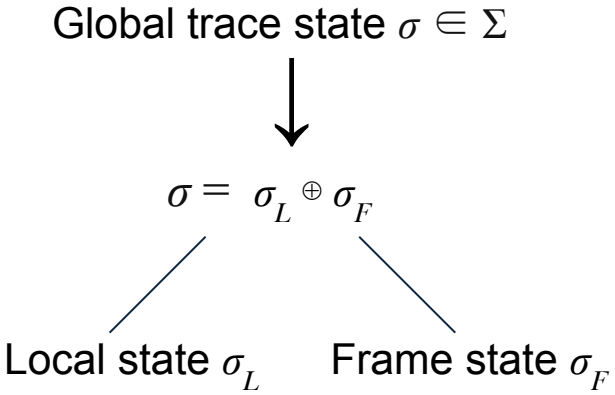
Example Program

```
step : c_state => c_state
step σ =
  σ(count := count σ + 1)

prog =
  whileLoop T
    (λ_. modify_commit step) ()
```

SL Over Trace Monad

Trace monad $f : \Sigma \to \mathcal{P}(\text{Trace} \times \text{Out}(\alpha))$



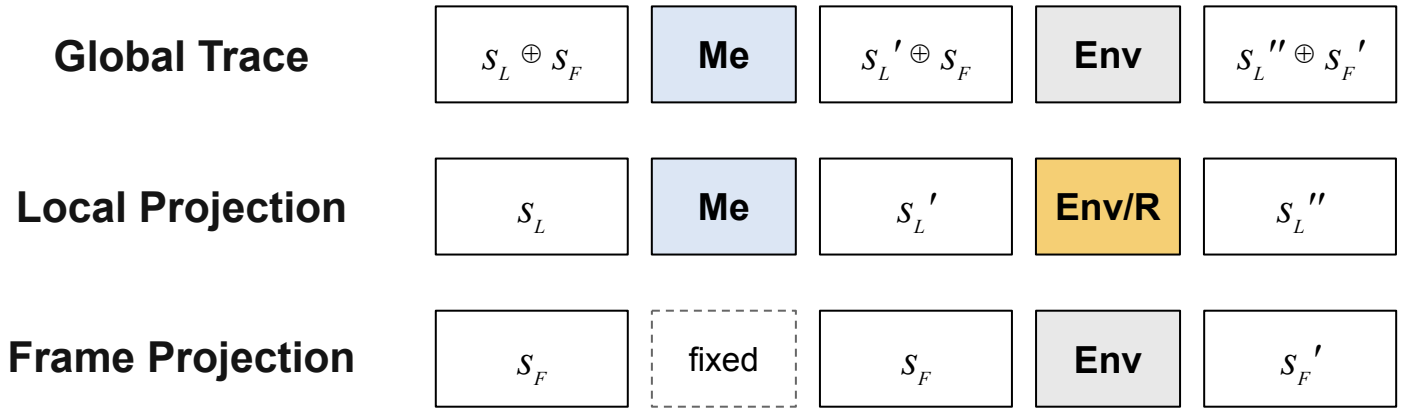
Σ has:

ε	empty
$\sigma_1 \# \# \sigma_2$	disjoint
$\sigma_1 \oplus \sigma_2$	merge

Same trace semantics; each state can now be viewed as local $\oplus$ frame

Trace Splitting

A global run is split into a local projection and a frame projection.



Separating and Fencing Actions

Action separating conjunction $*_A$

$$\frac{\sigma = \sigma_L \oplus \sigma_F \qquad \sigma' = \sigma'_L \oplus \sigma'_F \qquad R_L \;\; \sigma_L \, \sigma'_L \qquad R_F \;\; \sigma_F \, \sigma'_F}{(R_L *_A R_F)\ (\sigma, \sigma')}$$

$*_A$ splits a transition into local + frame steps

Fenced Action $\triangleright$

$$\frac{I \text{ precise} \qquad R\ \sigma\ \sigma' \Rightarrow I\ \sigma \wedge I\ \sigma' \qquad I\ \sigma \Rightarrow R\ \sigma\ \sigma}{I \triangleright R}$$

$\triangleright$ makes “acts only on this resource” precise

RG Frame Rule

$$\frac{\{P\}\ \{R\}\ f\ \{G\}\ \{Q\}\quad \text{locality}(I, R, R', F, f)\quad I \triangleright R\quad \text{sta}(I, G)\quad \text{sta}(F, R')\quad P \leq I}{\{P * F\}\quad \{R *_{\text{A}} R'\}\ f\ \{G *_{\text{A}} G'\}\quad \{Q * F\}}$$

Locality: f doesn't "touch" other resources

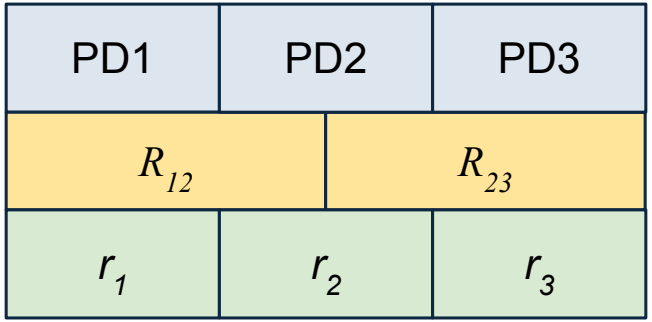
Every allowed global run of f from $s_L \oplus s_F$ (allowed by the global rely $R *_{\text{A}} R'$) splits into:

- a local run of f from s_L (allowed by R)
- a frame trace preserving F (allowed by R')

Case Study: 3-PD Microkit Pipeline

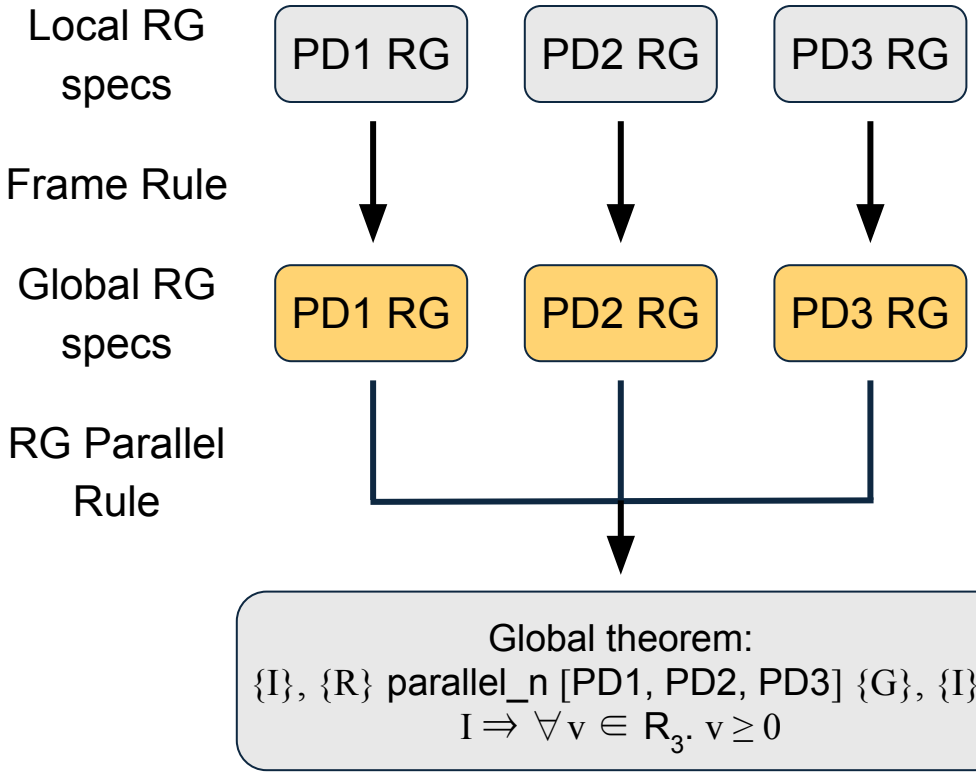
PD1: producer - produces arbitrary integers
PD2: filter - forwards only $v \geq 0$
PD3: consumer - appends to private log

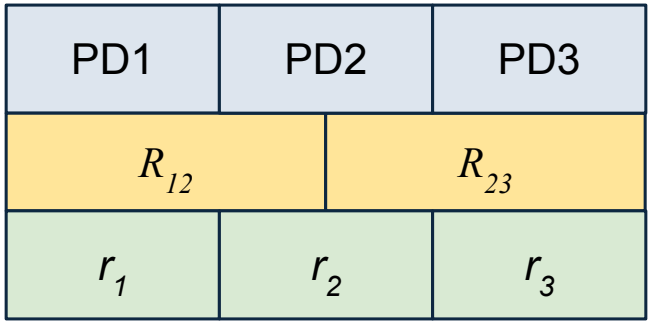
R_{12} mailbox: arbitrary int
R_{23} mailbox: non-negative int



Goal: $\forall v \in r_3.\ v \geq 0$

Case Study: 3-PD Microkit Pipeline





Conclusion & Future Work

Conclusion

- Mechanised LRG for non-deterministic trace monad semantics
- Minimalistic logic framework for Microkit purposes
- More detailed work in Isabelle Workshop informal proceeding

Future Work

- Connect local RG proof to ATP like Viper
- Clearer interface template & proof obligations for user
- Nested parallel combinator
- More realistic Microkit Examples

Thank You!
Questions?